

A Case Study in Object Oriented Modeling, Architecting, and Designing an Enterprise Monitoring Application

Sushil J. Louis

Genetic Algorithms Systems Lab
University Of Nevada, Reno
sushil@cs.unr.edu

John McDonnell, Doan Hohmeyer, Lisa
Heinselman, Andrew Walker
SPAWAR Systems Center
San Diego, CA
mcdonn@spawar.navy.mil

Abstract

We describe the use of object oriented techniques for the specification, architecting, and design of an enterprise monitoring application. Monitoring applications provide situational awareness and monitor aspects of an enterprise's activities. Drawing from a wide variety of static and dynamic data sources, they typically allow a user to specify items of interest, and drill down to obtain real-time or near-real time information on such items. Our paper describes the use of standard UML for modeling and the issues in architecting and designing our J2EE framework based application.

1. Introduction

Enterprise monitoring also known as business process monitoring is a subset of business process management applications. The goal is to allow members of an enterprise to obtain a view of event flow within the enterprise and to and from external entities. This view is usually tailored to the particular member's interests and needs. Many issues impact the modeling and design of business process monitoring systems, in particular, we dealt with

- Multiple static data bases
- Multiple dynamic data sources
- Multiple information update rates, and
- Asynchronous communication

An important trend in current enterprise application design is to use enterprise application frameworks such as .Net and J2EE for developing scalable, reliable, and secure enterprise applications. We chose the J2EE framework for these reasons and, in particular, for the J2EE framework's open standards and open source availability of J2EE application servers.

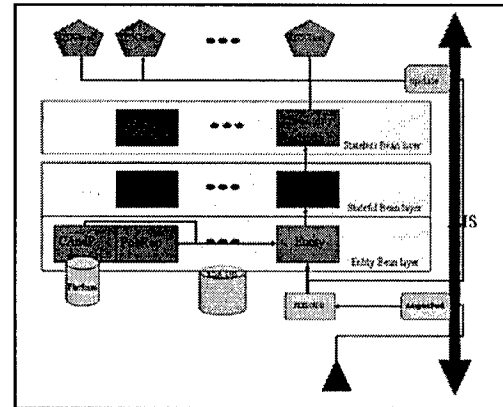


Figure 1: Mapping SIREN's architecture onto the J2EE framework.

Figure 1 depicts how our architecture maps into the three-tier J2EE application framework. Information flows from the bottom of the diagram (data producers) towards the top (data consumers) through entity, stateful session, and stateless session enterprise java beans. The triangle adaptor interfaces to a data source and asynchronously updates a Java Message Service (JMS) topic. The entity bean layer

20060425020

then persists this updated information in a database and relates this data with existing tables in the database. The stateful session bean layer provides a façade to the database for upper level clients. The stateless session bean layer provides a façade to presentation layer clients. The system can scale to handle hundreds of messages per second while providing real-time updates to dozens of clients.

Rather than delving into the details of a proprietary system, the main focus of this work is on how UML supports the modeling of our system within the J2EE framework [1, 2]. Many object oriented modeling and design methods appeared in the 90s and Booch, Rumbaugh, and Jacobson unified the various approaches into a standardized notation for design and analysis – the Unified Modeling Language [3]. UML provides a rich set of modeling tools to describe dynamic and static aspects of the system being designed. This case study describes our experience in using UML to model the SIREN situational awareness tool. We followed a spiral development model and made extensive use of UML for modeling and for design.

The next section describes the J2EE framework, the SIREN tool, and how we mapped SIREN requirements to the J2EE framework. The section uses UML diagrams to communicate different aspects of the system and our experience in using UML for requirements elicitation and feature specification. The last section concludes with the lessons learned from this project.

2. SIREN and the J2EE framework

The Java2 Platform, Enterprise Edition (J2EE) defines a standard for developing multitier enterprise applications. J2EE simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application

behavior automatically, without complex programming. Multitier scalable, reliable, and portable applications are hard to architect because they require bringing together a variety of skill-sets and resources, legacy data, and legacy code. In today's heterogeneous environment, enterprise applications have to integrate services from a variety of vendors with a diverse set of application models and other standards. Industry experience shows that integrating these resources can take up to 50% of application development time [5].

Security, messaging, transaction monitoring, and integration with legacy enterprise applications are all simplified within this enterprise application development framework. By providing this enterprise infrastructure, the framework allows for quick development, deployment, portability, and scalability.

SIREN is an enterprise state monitoring application that siphons data from a number of databases and data feeds, persists this data in its database, and feeds this organized data to client users. Clients specify the information that they would like to see. We use the application server's asynchronous messaging system to move data around this distributed system. Data sources are distributed over multiple servers on a network; each data source type has different control and data interfaces. Asynchronous communications among the components of this system make the system more robust.

2.1 Attaching an Adaptor

We use “adaptors” to connect to these different data sources and provide the server a façade to the underlying data source.

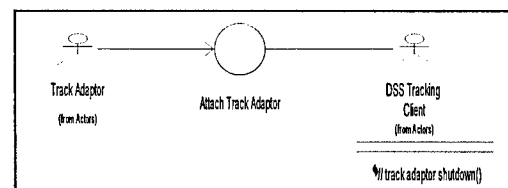


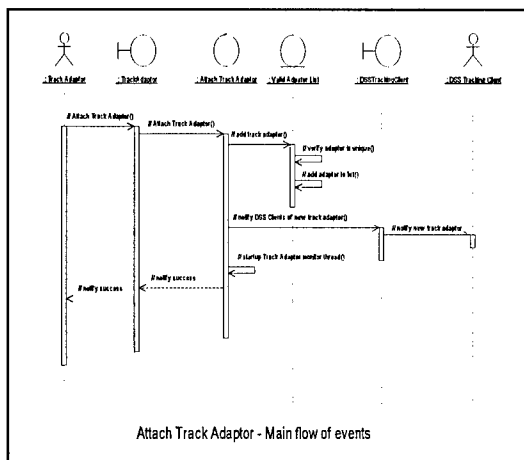
Figure 2: Use case for attaching a track adaptor.

```

    graph TD
      Start(( )) --> Wait[Waiting for new track adaptor]
      Wait --> Check[Make sure Track Adaptor signature is unique]
      Check -- "(Duplicate Signature)" --> NotifyFail[Notify Track Adaptor that registration failed because name was not unique]
      Check -- "(Unique Signature)" --> Add[Add Track Adaptor to list of valid adaptors]
      Add --> NotifyClients[Notify all DCS Tracking clients about the new Track Adaptor]
      NotifyClients --> StartThread[Start up Track Adaptor monitor thread]
      StartThread --> Monitor[Monitor Track Adaptor that registration was successful and accepted]
      Monitor --> Wait
  
```

The diagram illustrates the process of attaching a track adaptor. It begins with a start node leading to a stateful activity 'Waiting for new track adaptor'. A solid arrow leads to 'Make sure Track Adaptor signature is unique'. From here, a dashed arrow labeled '(Duplicate Signature)' leads to 'Notify Track Adaptor that registration failed because name was not unique', which then loops back to the start. A solid arrow labeled '(Unique Signature)' leads to 'Add Track Adaptor to list of valid adaptors'. This is followed by 'Notify all DCS Tracking clients about the new Track Adaptor', then 'Start up Track Adaptor monitor thread', and finally 'Monitor Track Adaptor that registration was successful and accepted'. A solid arrow from the monitor state loops back to the 'Waiting for new track adaptor' state. A note box explains that the monitor thread checks for unique signatures every 10 seconds and marks non-functional adaptors as nonfunctional in the 'valid adaptors' list.

The adaptor sends the server a request-to-attach message on the message bus. The server authenticates the adaptor, sets up a communication channel and notifies all clients of the new adaptor. This sequence of events is best denoted in the sequence diagram in Figure 4.



Once the adaptor has registered with the server, it obtains a JMS topic and begins sending information updates to that topic. These messages trigger updates to the underlying database and corresponding

[illegible]

Figure 5: Interfaces for the track adaptor

The server use case model below (Figure 6) depicts the overall operation of the server with respect to adaptor interactions.

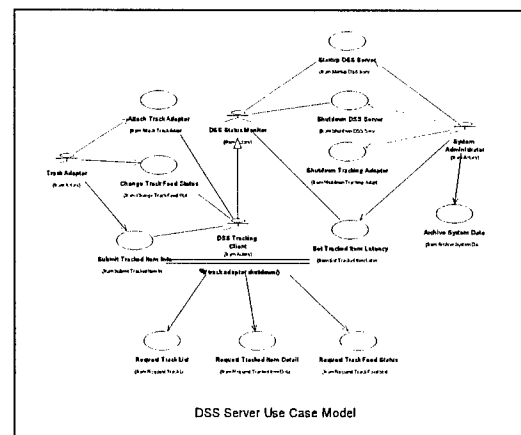
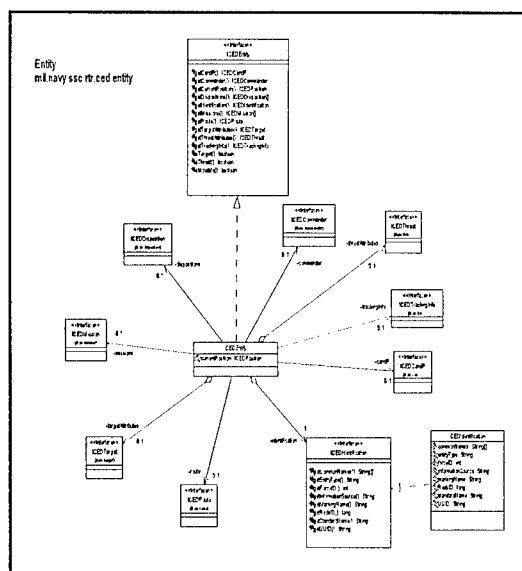


Figure 6: The Server Adaptor interaction use case model.

The class diagram for an entity class is given below (Figure 7). Most *getXXX* methods correspond to getting the contents of a column in the selected row of the entity database table. Some *getXXX* methods correspond to relations and return rows in other related tables in the database. The entity class therefore serves as the superclass for all objects of interest in the system. One interesting design debate arose over whether to transport the relatively heavyweight entity object over the network to all clients; our eventual solution was to transport only those portions of the entity requested by the client. The entity's location and type were always transported over the network while other entity data was made available as requested.

An entity object is stored in the server database as a collection of rows in multiple tables. Essentially we had to create a set of database tables and relations that distributed an entity's contents over these tables and relations. Commercial database vendors have spent years optimizing their product's performance and scalability. Mapping the complex entity object to a set of relational tables, allows our server database to optimize its storage and indexing so that database access is fast, transactional, and reliable. In contrast, if we had stored an entity object as a blob, a serialized object, we would not be taking advantage of commercial database optimizations. The **object-relational** mapping to map an entity to a set of relational database tables was done manually and was an important component of the final design. This up-front work in establishing a mapping allows scaling the system to handle high message (input) and client (output) loads.



2.3 Shutting down an adaptor

```

graph LR
    SA[System Administrator  
(Team Actor)] --- STA[Shutdown Tracking Adaptor  
(Team Actor)]
    STA --- DSS[DSS Tracking Client  
(Team Actor)]
    STA --- TA[Track Adaptor  
(Team Actor)]
    DSS --- TA
    STA -.->|!! track adaptor shutdown!| TA
  
```

Figure 8: Shutdown track adaptor use case.

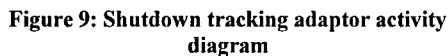
domain experts and confirm our understanding of the system’s requirements.

The UML modeling was used to obtain and confirm the design team's understanding of SIREN's requirements. However, while more detailed models were being acquired from end-users and domain experts, the architecture team used its broad knowledge of requirements to design and implement an end-to-end skeleton of the system. Skeleton development had three objectives:

- We had a heterogeneous development environment consisting of linux and windows machines. During testing the application server and database server ran on different windows machines with clients and adaptors distributed among windows and linux boxes. We used BEA's weblogic application server, Java, and MS SQLserver[6, 7, 8]. Although we found that our business logic was portable, porting to a different database (Oracle) was quite difficult because of differences in support for some primitive data types.

4. Conclusions

This paper described a case study in designing and implementing a large enterprise monitoring system. We found UML helped in acquiring and elucidating system specifications – both in dialogs with end users and domain experts and for



```

sequenceDiagram
    actor User
    participant Controller as Controller
    participant Model as Model
    participant View as View
    participant Service as Service
    participant Repository as Repository
    participant Domain as Domain

    User->>Controller: Find a book
    activate Controller
    Controller->>Model: Find a book
    activate Model
    Model->>Repository: Find a book
    activate Repository
    Repository->>Domain: Find a book
    activate Domain
    Domain->>Repository: Find a book
    deactivate Domain
    Repository->>Model: Find a book
    deactivate Repository
    Model->>View: Find a book
    activate View
    View->>Service: Find a book
    activate Service
    Service->>Controller: Find a book
    deactivate Service
    View->>Controller: Find a book
    deactivate View
    Controller->>User: Find a book
    deactivate Controller
    
```

The diagram illustrates the flow of a 'Find a book' request through various system components. The User initiates the process by sending a 'Find a book' message to the Controller. The Controller then passes this request to the Model, which in turn queries the Repository. The Repository interacts with the Domain to retrieve the book information. Once the information is retrieved, the Repository returns it to the Model, which then displays it to the View. The View sends the data to the Service, which returns it to the Controller. Finally, the Controller sends the result back to the User. A note specifies that the Controller is responsible for validating the input and that the Repository is responsible for returning the results.

More importantly, we have found that these three sets of UML diagrams, use-case, activity, and sequence diagrams, allow us to communicate clearly with end users and

programmers in refining and implementing the system. Developing an end-to-end architectural skeleton helped us gain a good understanding of implementation issues and get a feel for the system's scalability and reliability. Although we also expected the skeleton to be used as a template for further development and fleshing out of the system, we found that going through skeleton code was not as easy as going through UML and developers preferred to design and implement with little reference to the skeleton. Our next spiral development objective is to add functionality, test scalability, and bring the system up to CMM Level 3 [9].

5. Acknowledgements

This material is based upon work supported by the Office of Naval Research under contract number N00014-03-1-0104.

5. References

1. Grady Booch, Ivar Jacobson and James Rumbaugh. *The Unified Modeling Language User Guide*. Addison Wesley, 1998.
2. Sushil J. Louis, John McDonnell, and Nicholas Gizzi. "Dynamic Strike Force Asset Allocation Using Genetic Search and Case-Based Reasoning." In *Proceedings of SCI-02*, pages 423 - 428, July 2002.
3. John McDonnell, Nicholas Gizzi, and Sushil J. Louis. "Strike Force Asset Allocation Using Genetic Search." In Hamid Arabnia et al, editor, *Proceedings of the 2002 International Conference on AI, CSREA Press*, pages 897 - 901, June 2002.
4. Richard Monson-Haefel. *Enterprise JavaBeans (3rd Edition)*. O'Reilly and Associates, 2001.

5. Sun's J2EE website at <http://java.sun.com/j2ee/>.
6. BEA's website at <http://www.bea.com>
7. Sun's Java website at <http://java.sun.com>
8. Microsoft's SQLServer website at <http://www.microsoft.com/sql/default.asp>
9. The Capability Maturity Model webpage at <http://www.sei.cmu.edu/cmm/cmm.sum.html>